1. **MergeSort**:
    - The sorted lists are: [1, 16, 25, 31] and [-3, 0, 16, 27]
    - Compare 1 and -3 & -3 is smaller → [-3]
    - Compare 1 and 0 & 0 is smaller → [-3,0]
    - Compare 1 and 16 & 1 is smaller → [-3, 0, 1]
    - Compare 16 and 16 it is the same so merge 16 from the 1st list → [-3,0, 1, 16]
    - Compare 16 and 25; 16 is smaller so it chosen from the second list → [-3, 0, 1, 16, 16]
    - Compare 25 & 27; 25 is smaller → [-3, 0, 1, 16, 16, 25]
    - Compare 31 & 27; 27 is smaller → [-3, 0, 1, 16, 16, 25, 27]
    - The second list is done, so 31 from the first list is added → [-3, 0, 1, 16, 16, 25, 27, 31]
    - Final list: [-3, 0, 1, 16, 16, 25, 27, 31]

2. **InsertionSort:**
    - The unsorted list is: [-1, -5, 67, -10, 21, 8, 4, 1]
    - Compare -5 (second element) with -1 (first element)
    - -5 is smaller so shift -1 to the right and insert -5 at index 0 so it is the first element now
    - Compare 67 (third element) with -1
    - Do not need to shift 67 so it stays where it is
    - Compare -10 with 67
        - 67 > -10 so shift 67 right
        - -1 > -10 so shift -1 right
        - -5 > -10 so shift -5 right
        - Insert -10 at index 0 so it is the first element since it is the smallest
    - Compare 21 with 67 & since 67 > 27, shift 67 right
    - Insert 21 at index 3 because it is smaller than 67 but greater than -1
    - Compare 8 with 67, 67 > 8 so shift 67 right
    - Compare 8 with 21, 21 > 8 so shift 21 right
    - Insert 8 at index 3 because it is greater than -1 but smaller than 21
    - Compare 4 with 67, 67 > 4 so shift 67 right
    - Compare 4 with 21, 21 > 4 so shift 21 right
    - Compare 4 with 8, 8 > 4, so shift 8 right
    - Compare 1 with 67, 67 > 1 so shift 67 right
    - Compare 1 with 21, 21 > 1 so shift 21 right
    - Compare 1 with 8, 8 > 1, so shift 8 right
    - Compare 1 with 4, 4 > 1, so shift 4 right
    - Insert 1 at index 3 because it is greater than -1 but smaller than 4
    - Now the sorted list is: [-10, -5, -1, 1, 4, 8, 21, 67]

3. **Quicksort:**
    - Pivot = 19

- Partition the list by putting elements less than 19 on the left and elements greater than 19 on the right → [-5, 6, 11, -3] 19 [42, 25, 26]
- Left sublist: the middle element is 6 so it is the pivot
- Partition the list by putting elements less 6 on the left and elements greater than 6 on the right → [-5, -3] 6 [11]
- another sublist: pivot is -3, partition the list → -5 is on the left since it is less than -3 so the rearranged list is → [-5] -3
- Right sublist [42, 25, 26]: the middle element is 25 so that is the pivot
- When you partition the list, nothing on the left side because nothing is less than 25 so the new sublist is 25 [42,26]
- For [42,26], 26 is the pivot and the rearranged list is 26 [42]
- When you combine the sorted sublists & the pivot, the sorted list is [-5,-3,6,11,19,25,26,42]

4. **Shellsort:**
    - Unsorted list: [15, 14, -6, 10, 1, 15, -6, 0]
    - First gap = 4, which means to compare elements 4 positions apart
    - Compare 15 & 1 → swap → [1, 14, -6, 10, 15, 15, -6, 0]
    - Compare 14 & 15 → no swap → [1, 14, -6, 10, 15, 15, -6, 0]
    - Compare -6 & 6: no swap → [1, 14, -6, 10, 15, 15, -6, 0]
    - Compare 10 & 0: swap → [1, 14, -6, 0, 15, 15, -6, 10]
    - Divide gap in half so gap = 2 now which means to compare the elements 2 positions apart
    - Compare 1 & -6 → swap → [-6, 14, 1, 0, 15, 15, -6, 10]
    - Compare 14 & 0 → swap → [-6, 0, 1, 14, 15, 15, -6, 10]
    - Compare 1 & 15 → no swap
    - Compare 14 & -6 → swap → [-6, 0, 1, -6, 15, 15, 14, 10]
    - Divide gap in half so gap = 1 now which means to compare the elements like a regular insertion sort
    - Compare 0 and -6: No swap, Compare 1 and 0: No swap,
    - Compare -6 and 1: Swap → [-6, 0, -6, 1, 15, 15, 14, 10]
    - Compare 15 and 1: No swap
    - Compare 15 and 15: No swap
    - Compare 14 and 15: Swap → [-6, 0, -6, 1, 15, 14, 15, 10]
    - Compare 10 and 15: Swap → [-6, 0, -6, 1, 15, 14, 10, 15]
    - Compare 10 and 14: Swap → [-6, 0, -6, 1, 15, 10, 14, 15]
    - Compare 10 and 15: No swap
    - Sorted list: [-6, -6, 0, 1, 10, 14, 15, 15]

5. **Sorting Algorithms Ranking:**

    1. Mergesort → O(n log n) - In the best and worst-case scenario with any list, merge sort will split into two and recursively sort. Then the arrays would merge by adding the smallest element among the two arrays to a new array.

    2. Quicksort → O(n^2) - The worst case is O(n^2) when the pivot value results in empty left or right lists and has the maximum possible number of recursive calls

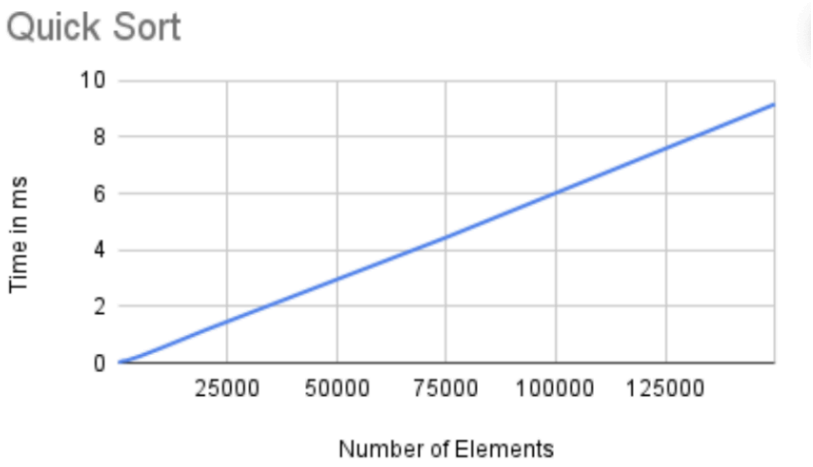
3. Shellsort - Shell sort uses gaps to sort and swap values, selection of gap increments is critical to performance of shell sort

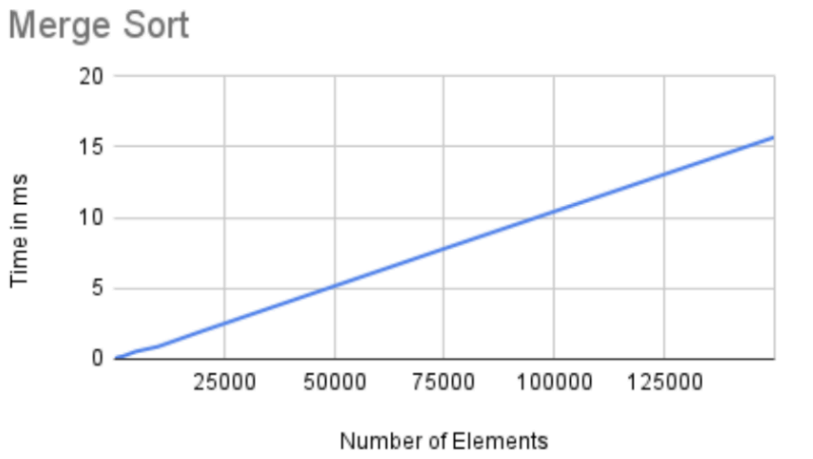
4. Insertion - Worst case scenario when the max possible number of swaps is reached since insertion sort removes element from unsorted

5. Bubble - Worst case scenario is when all numbers must be swapped because bubble sort compares neighboring elements to find which elements to swap

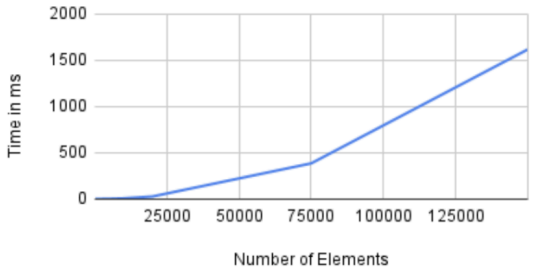
6.Selection - Worst case scenario is that the algorithms always does two passes through the array since it has to find the smallest element in the array and swap it to the index with the first unsorted element.

9.

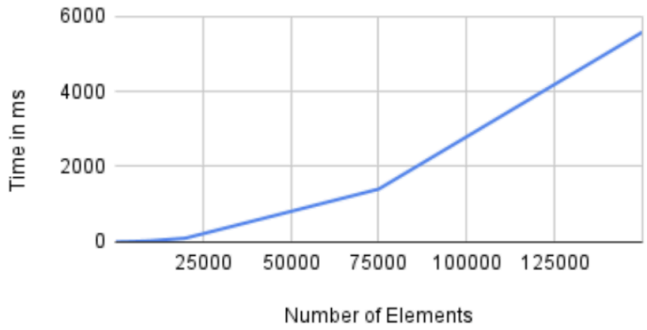




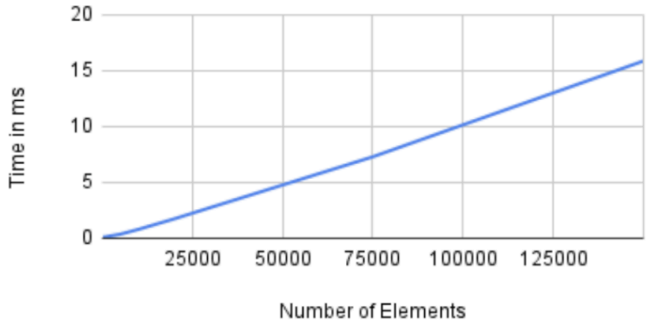
Insertion Sort



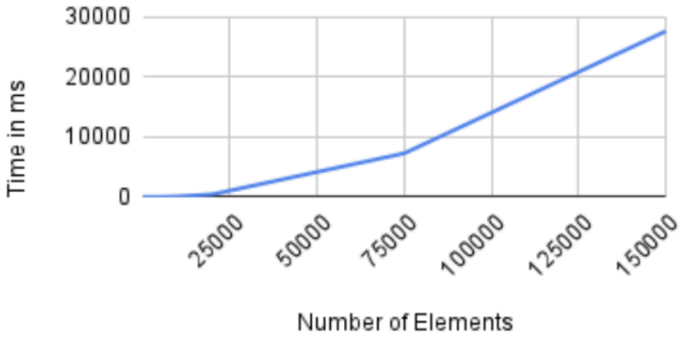
Selection Sort



Shell Sort

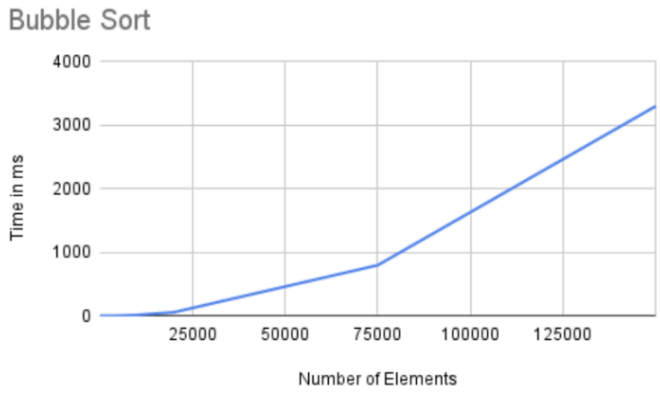


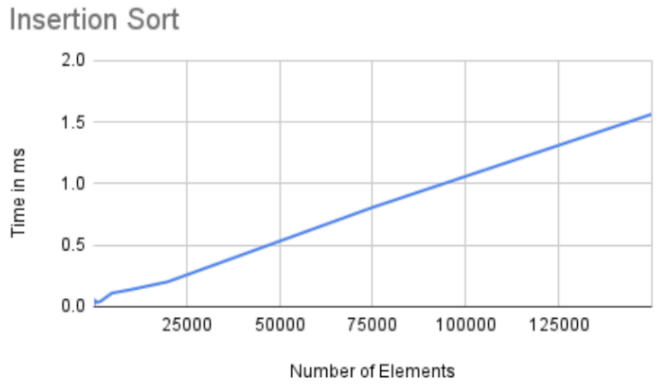
Bubble Sort



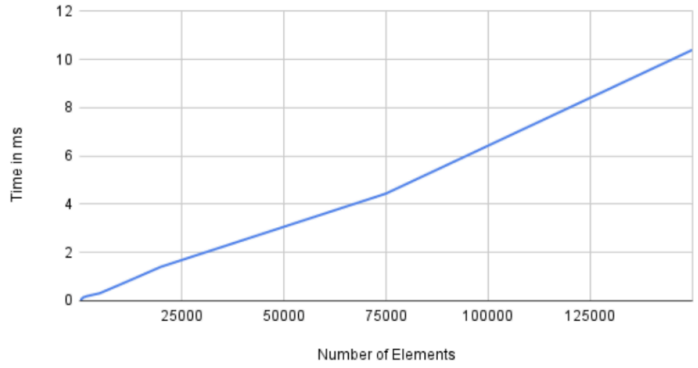
10. One thing I noticed about the relative performance of the different algorithms is that Bubble Sort performed the worst and Quick Sort performed the best. Bubble Sort took over 40 seconds to sort all 150,000 elements, whereas Quick Sort took around 15 milliseconds. Most of these results matched my ranking except for the fastest algorithm. I thought Merge Sort would be the best performance, however it was second to Quick Sort. I found this very interesting because they have the same time complexity of O(N log N). Bubble Sort is the most impractical for large datasets, since it was the slowest.

11. All of the algorithms revealed performance nearly identical to the initial tests—several algorithms even shaved off a few seconds. The only hiccup was that Quicksort occasionally triggered stack-overflow errors.
Graphs:

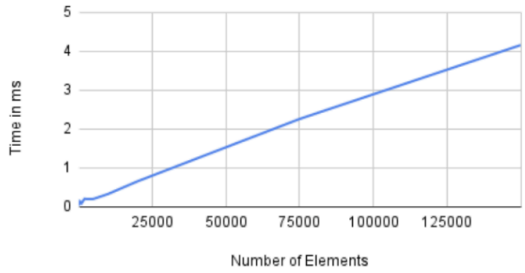




Merge Sort

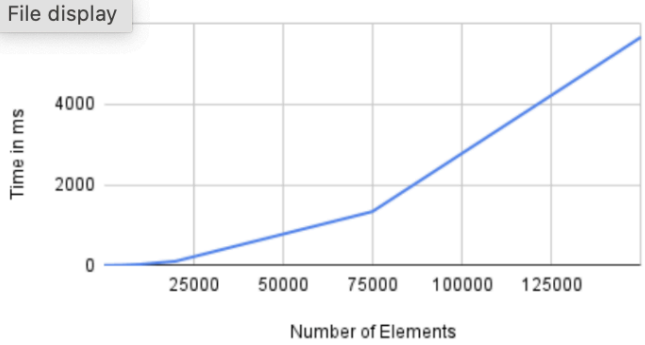


Shell Sort



Selection Sort

File display



Quick Sort